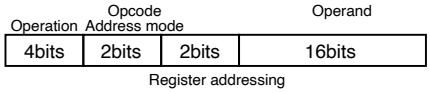
ALevel CS C06 Assembly Language

© 2021,有鱼英语
https://collapsar.yuketang.net/
https://coding.yuketang.net/

Machine code instructions

Facts:
1. The only language that the CPU recongnises is machine code
2. Machine code consists of a **sequence of instructions**
3. Different processors have different instruction sets associate with them

Opcode: defines the action associated with the instruction
Operand: defines any data needed by the instruction
Machine code instruction: a binary code with a defined number of bits that comprises an opcode and, most often, one operand



Symbolic addressing: The use of symbolic addressing allows a programmer to write some assembly language code without having to bother about where the code will be stored in memory when the program is run.
Relative addressing: Illustration identifying the offset from the base address which is the address of the first instruction in the program.
Absolute addressing: Identifying actual memory addresses

Addressing mode: when the instruction uses a value this defines how the operand must be used to find the value
Immediate addressing: The operand is **the value** to be used in the instruction. SUM #48, MOV AX,3064H
Direct addressing: The operand is **the address which holds the value** to be used in the instruction. ADD TOTAL, MOV AX, [2000H]
Indirect addressing: The operand is **an address that holds the address which has the value to be used** in the instruction. MOV AX, [BX]
Index addressing: The operand is **an address to which must be added the value currently in the index register (IX)** to get the address which holds the value to be used in the instruction. Mov AX, COUNT[SI]

Assembly language instruction groups: data movement, input and output, comparisons and jumps, arithmetic operations, shift operations.

Data movement: involve loading data into a register or storing data in memory.

opcode	openand	exmpanation
LDM	#n	Immediate addressing. Load the number n to **ACC**.
LDR	#n	Immediate addressing. Load the number n to **IX**.
LDD	<address>	**Direct addressing**. Load the **contents** at the given address to ACC.
LDI	<address>	**Indirect addressing**. The address to be used is at the given address. Load **the contents of this second address** to ACC.
LDX	<address>	**Indexed addressing**. Form the address from <address> + the contents of the index register. Copy **the contents of this calculated address** to ACC.
MOV	<register>	Move the contents of the accumulator to the given register (**IX**).
STO	<address>	Store the contents of **ACC** at the given address.

Arithmetic operations:
There are no instructions for general-purpose multiplication or devision.General-purpose addition and subtraction are created for

opcode	openand	exmpanation
ADD	<address>	Add the contents of the given address to the ACC
ADD	#n	Add the denary number n to the ACC
SUB	<address>	Subtract the contents of the given address from the ACC
SUB	#n	Subtract the denary number n from the ACC
INC	<register>	Add 1 to the contents of the register (ACC or IX)
DEC	<register>	Subtract 1 from the contents of the register (ACC or IX)

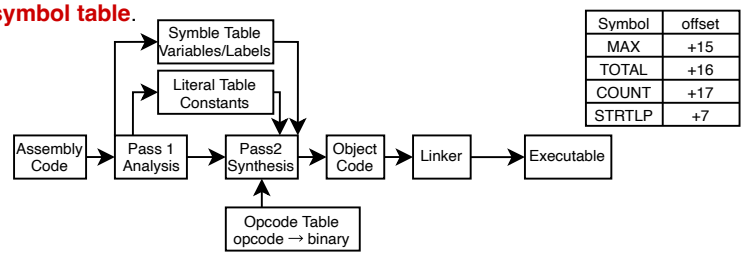
Shift operations:
Logical shift: where bits in the accumulator are shifted to the right or to the left and a zero moves into the bit position vacated
Cyclic shift: similar to a logical shift but bits shifted **from one end reappear at the other end**
Arithmetic shift: uses the shift to carry out multiplication or division of a **signed integer** stored in the accumulator

Assembly Language

Assembly language: a low-level language related to machine code where opcodes are written as mnemonics and there is a character representation for an operand
Assembler: a program used to translate an assembly language program into machine code

The essence of assembly language is that for each **machine code instruction** there is an equivalent **assembly language instruction** which comprises:
a mnemonic (a symbolic abbreviation) for the opcode
a character representation for the operand
Assembly language special features:
1. comments
2. symbolic names for constants
3. labels for addresses
4. **macros** (a sequence of instructions that is to be used more then once in a program)
5. **directives** (an instruction to the assembler program)

Two-pass assembler: design to handle programs written in the style of the one illustrated.
This program contains **forward references**. Some of the instructions have a symbolic address for the operand where the location of the address is not known at that stage of the program. A two-pass assembler is needed so that in the first pass the location of the addresses for **forward references can be identified**.
When a symbolic address is met for the first time its name is entered into the **symbol table**.



Input and Output:
1. **IN**: stored in the ACC the ASCII value of a character typed at the keyboard.
2. **OUT**: display on the screen the character for which the ASCII code is stored in the ACC

Comparisons and jumps:

opcode	openand	exmpanation
JMP	<address>	Jump to the given address
CMP	<address>	Compare the contents of ACC with the contents of <address>
CMP	#n	Compare the contents of ACC with the number n
CMI	<address>	**Indirect addressing**. The address to be used is at the given address. **Compare the contents of ACC with the contents of this second address**
JPE	<address>	**Following a compare instruction**, jump to <address> if the compare was **True**
JPN	<address>	**Following a compare instruction**, jump to <address> if the compare was **False**

Bitwise logic operation:

opcode	openand	exmpanation
AND	#Bn	Bitwise AND operation of the contents of ACC with the binary number n
AND	<address>	Bitwise AND operation of the contents of ACC with the contents of <address>
XOR	#Bn	Bitwise XOR operation of the contents of ACC with the binary number n
XOR	<address>	Bitwise XOR operation of the contents of ACC with the contents of <address>
OR	#Bn	Bitwise OR operation of the contents of ACC with the binary number n
OR	<address>	Bitwise OR operation of the contents of ACC with the contents of <address>

Shift operations:
1. LSL #n : where the bits in the accumulator are shifted logically n places to the left
2. LSR #n : where the bits are shifted to the right.